Symmetric Structure in Logic Programming

Jin-Zhao Wu\textsuperscript{1,2} and Harald Fecher\textsuperscript{2}

\textsuperscript{1}Chengdu Institute of Computer Applications, The Chinese Academy of Sciences, Chengdu 610041, P.R. China
\textsuperscript{2}Fakultät für Mathematik und Informatik, Universität Mannheim, D-7, 68131 Mannheim, Germany

E-mail: wu@ipil.informatik.uni-mannheim.de

Received February 28, 2002; revised February 16, 2004.

Abstract It is argued that some symmetric structure in logic programs could be taken into account when implementing semantics in logic programming. This may enhance the declarative ability or expressive power of the semantics. The work presented here may be seen as representative examples along this line. The focus is on the derivation of negative information and some other classic semantic issues. We first define a permutation group associated with a given logic program. Since usually the canonical models used to reflect the common sense or intended meaning are minimal or completed models of the program, we expose the relationships between minimal models and completed models of the original program and its so-called G-reduced form newly-derived via the permutation group defined. By means of this G-reduced form, we introduce a rule to assume negative information termed G-CWA, which is actually a generalization of the GCWA. We also develop the notions of G-definite, G-hierarchical and G-stratified logic programs, which are more general than definite, hierarchical and stratified programs, and extend some well-known declarative and procedural semantics to them, respectively.

Keywords symmetry, logic programming, semantics

1 Introduction

In general, logic programming does not take advantage of symmetries. However, symmetric structures in logic programs play a role and are worthwhile to be explored further. This paper represents some initial work in this regard. The results presented here may be seen as representative examples to demonstrate how to utilize symmetric structures in a logic program as a useful factor in logic programming. The focus is on the derivation of negative information and some other classic model and procedural semantics.

Two criteria to evaluate a given semantics are its declarative ability to reflect the common sense or intended meaning and the expressive power of the language characterized by it. We show that the introduction of a symmetric structure in a logic program may enhance, e.g., the declarative ability of GCWA and the expressive power of the SLD-resolution, completion procedure, SLDNF-resolution as well as the standard model semantics.

The GCWA\textsuperscript{[1]} does not allow any elementary facts in the minimal models to be negative. Therefore, the corresponding declarative semantics sometimes fails to reflect the common sense or intended meaning of the program. We present, through taking a symmetric structure into consideration, a proposal termed G-CWA to deal with negative information, which is more powerful from the viewpoint of declarative ability. It is actually a generalization of the GCWA.

The SLD-resolution\textsuperscript{[2]}, SLDNF-resolution\textsuperscript{[3]}, and the standard model (iterative fixed-point) semantics\textsuperscript{[4]} are effective for definite, hierarchical and stratified programs, respectively. The completion procedure\textsuperscript{[5]} may lead to inconsistency for non-stratified programs. Therefore, they are restrictive from the point of view of expressive power. Through taking a symmetric structure into consideration, we define three classes of logic programs termed G-definite, G-hierarchical and G-stratified programs, which are more general than definite, hierarchical and stratified programs. Furthermore, we extend the SLD-resolution, the SLDNF-resolution as well as the standard model semantics to these programs, and show that the completion of a G-stratified program is consistent.

To this end, a great deal of different work has been done in the literature. The careful CWA and extended CWA\textsuperscript{[6]} generalized the GCWA. The perfect model semantics for locally stratified programs\textsuperscript{[7]} generalized the standard model semantics. After showing the completeness result of SLDNF-resolution in a standard textbook\textsuperscript{[8]}, the author declared that the completeness of SLDNF-resolution was of such importance that finding more general completeness results was an urgent
priority. Later on, this completeness result was extended to the cases of call-consistent and strict stratified programs\cite{9,10}. Moreover, it was shown in \cite{9} that the completion of a call-consistent program is consistent.

On other sophisticated formalizations and extensions, the reader may consult \cite{11-14,15-17}. [15-17] are thorough and comprehensive survey papers in this regard. In this paper, we do not involve ourselves in circumscriptions\cite{9} and the semantics using non-classical logics (See \cite{17} and the references therein). We concentrate on the classical first-order logic framework.

Remark that our work on various model and procedural semantics is not covered by any aforementioned results. This is actually not theoretically important. The point is that a methodology is proposed. We can similarly consider, for example, by taking a symmetric structure into account, the careful CWA, various semantics for strict, locally stratified, and call-consistent programs. So, one purpose of this paper is to suggest taking symmetric structures as an involved factor when developing semantics.

2 Basic Knowledge

Some basic notions are described on which our following discussions are based.

2.1 Permutation Groups

Let \( R \) be a non-empty set. A permutation (on \( R \)) is a bijection from \( R \) to \( R \). The set \( S_R \) of all the permutations on \( R \) forms a group under the operation of function composition, and is usually called the symmetric group (on \( R \)) when \( R \) is finite. We call the subgroups of \( S_R \) permutation groups.

Let \( G \) be a permutation group on \( R \). We define a relation \( \sim \) on \( R \) by the rule: For \( r_1, r_2 \in R \), \( r_1 \sim r_2 \) iff there exists \( \sigma \in G \) such that \( \sigma(r_1) = r_2 \). \( \sim \) is an equivalence on \( R \). Its equivalence classes are called orbits of \( G \).

Suppose that \( O_1, \ldots, O_m \) constitute a partition of \( R \). \( O_i \) is said to be proper if it contains more than one element. For any \( \sigma_1 \in S_{O_1}, \ldots, \sigma_m \in S_{O_m} \), let \( \sigma \) be the following permutation on \( R \):

\[
\sigma(r) = \sigma_i(r), \quad \text{if} \quad r \in O_i.
\]

By \( G = S_{O_1} \times \cdots \times S_{O_m} \) we denote the set of all such permutations. \( G \) is then a subgroup of \( S_R \), and its orbits are \( O_1, \ldots, O_m \). We use \( Fix(G) \) to denote the union of those \( O_i \) that are not proper. It consists of all the elements fixed by \( G \).

We refer the reader to \cite{18} for more details on permutation groups.

2.2 Logic Programming

For the basic notions we use but do not define in this paper like atoms, formulas, interpretations as well as minimal models, we refer the reader to \cite{8}.

To be clear enough, for the major part of this paper we are only concerned with the propositional case. All the results are generalized to the first-order predicate case in the end of the paper.

A (normal) clause is a formula of the form

\[ (\land_{i=1}^n A_i) \land (\land_{j=1}^m (\neg B_j)) \rightarrow B_w, \]

where \( A_i, B_j, B_w \) are atoms. If \( v = 0 \), we call this clause definite.

For literals \( L_1, \ldots, L_k, L_1 \land \cdots \land L_k \rightarrow \) is called a (normal) goal. It is said to be definite if \( L_1, \ldots, L_k \) are all positive.

A (normal logic) program is a finite set of clauses. It is called definite if all its clauses are definite.

Let \( P \) be a program. By the P-definition of an atom \( r \), we mean the subset of \( P \) consisting of all clauses with \( r \) as the heads. \( P \) is called stratified [hierarchical], if there exists a partition

\[ P = P_1 \cup \cdots \cup P_m \]

such that 1) \( P_1 \) can be empty and \( P_i \cap P_j = \emptyset (i \neq j) \); 2) if an atom occurs positively in the body of a clause in \( P_i \), then its P-definition is contained in \( \cup_{j<i} P_j \) [resp. \( \cup_{j<i} P_j \) ]; 3) if an atom occurs negatively in a clause in \( P_i \), then its P-definition is contained in \( \cup_{j<i} P_j \). Clearly, definite and hierarchical programs are stratified.

Suppose that \( r \) is an atom. If the P-definition of \( r \) is empty, we say that the formula \( \neg r \) is the completed P-definition of \( r \). Otherwise, let \( \land_j L_{ij} \rightarrow r \) be all the clauses in the P-definition of \( r \). We call the formula \( \forall i (\land_j L_{ij}) \leftrightarrow r \) the completed P-definition of \( r \). The completion of \( P \), denoted by \( Comp(P) \), is the collection of completed P-definitions of atoms.

By \( S \models F \) we mean the formula \( F \) follows from the formula set \( S \). Namely, the models of \( S \) are models of \( F \). The models of the completion \( Comp(P) \) of program \( P \) are called the completed models of \( P \).

2.3 Notations

Throughout the paper, \( P \) stands for a normal program, and \( R \) the set of all atoms appearing in
\[ P. \text{ As usual, an interpretation } I \text{ is represented by } I = \{ r \mid I(r) = 1 \}, \text{ where } r \text{ is an atom.} \]

For an expression \( T \), we use \( \sigma(T) \) to denote the expression obtained from \( T \) by substituting any atom \( r \) occurring in \( T \) by \( \sigma(r) \). For \( G \subseteq S_R \), \( G(T) = \{ \sigma(T) \mid \sigma \in G \} \). Obviously, \( P \) is stratified (hierarchical, definite) iff \( \sigma(P) \) is stratified (resp. hierarchical, definite).

For a literal \( L \) and a literal set \( M \), let \( L^n = \wedge_{i=1}^n L \) (\( L^0 = 1 \)), where \( n \) is a natural number, and \( \neg \) be the set consisting of the atoms occurring in the positive [resp. negative] literals in \( M \).

For a clause \( C = (\wedge_{i=1}^u A_i) \land (\wedge_{j=1}^v \neg B_j) \rightarrow B_w \), \( M_C \) denotes the set consisting of \( \neg A_i \), \( B_j \) and \( B_w \) (\( i = 1, \ldots, u \), \( j = 1, \ldots, v \)).

\( M_C \) actually represents the disjunction form of \( C \) without any atom ordering, and the numbers of occurrences of literals are neglected. Bearing in mind this disjunction form, we see that it is natural that \( M_C \) consists of all the atoms occurring positively in the body of \( C \), whereas \( M_C \) consists of all the atoms occurring negatively in the body as well as the head of \( C \).

When we write \( C \) in the form
\[ C = (\wedge_{i=1}^u A_i^{a_i}) \land (\wedge_{j=1}^v \neg B_j^{b_j}) \rightarrow B_w \]
we always suppose that \( a_i, b_j \) are natural numbers, \( B_w \) is an atom, \( A_i \) are atoms different from each other, and so are \( B_j \).

### 3 Symmetric Structures

We define a permutation group \( G \) associated with \( P \), and construct a new program called \( G \)-reduced form of \( P \). All the following definitions are based on these notions. We present as well two examples to illustrate that there exist such theoretically non-trivial permutation groups.

#### 3.1 Definition

Let \( O_1, \ldots, O_m \) constitute a partition of \( R \). We say that for \( P \) this partition is well-arranged if for any \( C \in P \) the following condition is met:

If \( M_C^r \cap O_i \neq \emptyset \) for some proper \( O_i \), then there exist \( O_j \) and two different \( r_1, r_2 \in O_j \), such that \( r_1, r_2 \in M_C^r \); otherwise, for an arbitrary \( r_1 \in O_1 \), then \( O_1 \subseteq M_C^r \).

If for \( P \), \( O_1, \ldots, O_m \) constitute a well-arranged partition of \( R \), then we call \( G = S_{O_1} \times \cdots \times S_{O_m} \) a permutation group associated with \( P \).

The main reason why we define \( G \) is that we need the action of \( G \) to resume exactly the minimal or completed models of \( P \). Yet, when we mention \( G \) the corresponding well-arranged partition for \( P \) just consists of all the orbits of \( G \). It associates with \( G \) naturally.

#### 3.2 \( G \)-Reduced Forms

We observe that two atoms in the same orbit of \( G \) cannot simultaneously appear in the same minimal model of \( P \). This is also the case for the complete models of \( P \) when \( P \) is hierarchical. We can thus remove some clauses and literals from \( P \) without losing useful information. The following is a procedure to reduce the symmetry and create a new program. From now on, \( G \) represents a permutation group associated with \( P \), and \( O_1, \ldots, O_m \) are all the orbits of \( G \).

First, in each orbit \( O_k \) of \( G \) choose an atom \( r_k \), and call it the representative of this orbit. Let \( C(0) = C \); for \( k \geq 1 \), assume
\[ C(k-1) = A \land B \rightarrow B_w, \]
\[ A = \lambda_{i=1}^u A_i^{a_i}, \]
\[ B = \lambda_{j=1}^v \neg B_j^{b_j}. \]

If there exists \( A_i \), such that \( A_i \in O_k \) and \( O_k \) is proper, then let \( E(C) = \emptyset \); otherwise, let \( E(C) = \{ C(m) \} \), where \( C(m) \) is the clause derived as follows.

Suppose that \( B_{j_1}, \ldots, B_{j_s} (\neq \emptyset) \) are all the atoms occurring in \( B \) such that each \( B_{j_i} \in O_k \). Let
\[ B_{O_k} = \wedge_{j \in J \setminus \{ j_1, \ldots, j_s \}} \neg B_j^{b_j}, \]
where \( J = \{ j_1, \ldots, j_s \} \), \( V = \{ 1, \ldots, v \} \). Obviously \( J \subseteq V \). For \( r \in R \) define
\[ b(r) = \begin{cases} 1, & \text{if } \exists j : B_{j} = r; \\ 0, & \text{otherwise}. \end{cases} \]

If \( B_w \not\in O_k \), then
\[ C(k) = A \land (B_{O_k}) \land (\neg r_k) \rightarrow B_w. \]
If \( B_w \in O_k \), then
\[ C(k) = A \land (B_{O_k}) \land (\neg r_k)^{b_k} \rightarrow r_k. \]

Let \( G(P) = \bigcup_{C \in P} E(C) \), and call it a \( G \)-reduced form of \( P \).

In this procedure, some clauses and literals are removed from \( P \) when \( G \) is not trivial. Notice that \( G \) and \( G(P) \) can analogously be defined in the case when \( P \) is a general clause (a disjunction of literals) set. In fact, a disjunction of negative atoms can be treated as a clause with an empty head. For the notions independent of syntax like minimal models, we may suppose that \( P \) is a consistent general
clause set instead of a normal program. But for syntax-sensitive semantics like the completion procedure or the standard model semantics, we have to take into account the numbers of occurrences of some literals, and which atom is chosen as the head of a clause is very important.

Let $P_\alpha = \{ C \in P \mid M_C ^\alpha \subseteq \text{Fix}(G) \}$. For any proper orbit $O$ of $G$, if there exist $r \in O$ and $C \in P - P_\alpha$ such that $r$ is the head of $C$, then in the procedure to form $G(P)$, choose this $r$ as the representative of $O$; otherwise, choose an arbitrary element of $O$ as its representative. Such a $G(P)$ is said to keep heads.

3.3 Examples

The trivial, namely the identity, permutation group on $R$ is clearly a permutation group associated with $P$. Here, we present two theoretically non-trivial examples. For more details about them, please see [19].

3.3.1 Disjunction Invariant Permutation Group

For $C \in P$, let

$$G_C = \{ \sigma \in S_R \mid \sigma(M_C) = M_C \}.$$  

$G_C$ consists of all the permutations keeping invariant the disjunction form of $C$. It is a permutation group on $R$. Let

$$G = \cap_{C \in P} G_C.$$  

$G$ is again a permutation group on $R$. All the disjunction forms of the clauses in $P$ are preserved under $G$. We call $G$ the disjunction invariant permutation group of $P$.

Let $O_1, \ldots, O_m$ be all the orbits of $G$. Then it is not hard to see that for $P$, $O_1, \ldots, O_m$ constitute a well-arranged partition of $R$, and $G = S_{O_1} \times \cdots \times S_{O_m}$. Therefore, $G$ is a permutation group associated with $P$.

3.3.2 Sequential Disjunction Invariant Permutation Group

We construct a “bigger” (in contrast to the size of orbits) permutation group associated with $P$ from a sequence of the disjunction-invariant permutation groups defined above.

Let $P_0 = P$, and $G_0$ the disjunction-invariant permutation group of $P$. For $k \geq 0$, while $P_k \neq \emptyset$ and $G_k$ is non-trivial, let $P_{k+1}$ be a $G_k$-reduced form of $P_k$, and $G_{k+1}$ the disjunction-invariant permutation group of $P_{k+1}$ if $P_{k+1} \neq \emptyset$.

This procedure terminates since when $G_k$ is not trivial the number of elements of $P_{k+1}$ is less than that of $P_k$. Upon the termination, we obtain a permutation group sequence $G_0, \ldots, G_n$. Let

$$O = \bigcup_{k=0}^n \{ O \mid O \text{ is an orbit of } G_k \}.$$  

We define a relation $\leftrightarrow$ on $O$:

$$O, O' \in O, O \leftrightarrow O' \text{ iff } O \cap O' \neq \emptyset.$$  

$\leftrightarrow$ is reflexive and symmetric. Let $\leftrightarrow^*$ be its transitive closure. $\leftrightarrow^*$ is then an equivalence on $O$. Let $O_i (i = 1, \ldots, m)$ be all the equivalence classes of $\leftrightarrow^*$, and $O_i = \cup_{O \in O} O$. Then $O_i (i = 1, \ldots, m)$ constitute a partition of $R$ which is well-arranged for $P$. Thus $G = S_{O_1} \times \cdots \times S_{O_m}$ is a permutation group associated with $P$, and is called the sequential disjunction invariant permutation group of $P$.

4 Minimal and Completed Models

The minimal and completed models of programs play a central role in logic programming. We demonstrate here the relationships between the minimal models, as well as the completed models, of $P$ and $G(P)$.

Lemma 4.1. Suppose that $O$ is an orbit of $G$, and $I$ a minimal model of $P$. For two different $r_1, r_2 \in O$, if $r_1 \in I$ then $r_2 \notin I$.

Proof. Otherwise, suppose that $O_i$ are all the orbits of $G$ whose intersections with $I$ contain more than one element. Choose $r_i \in O_i \cap I$. Since for $P$ the orbits of $G$ constitute a well-arranged partition of $R$, $I - (\cup_i (O_i - \{ r_i \})) \subset I$ is a model of $P$. This is in contradiction with the fact that $I$ is minimal. □

Lemma 4.2. (1) If $I_G$ is a minimal model of $G(P)$, then $I_G$ is a minimal model of $P$; (2) if $I$ is a minimal model of $P$, then for any $\sigma \in G$, $\sigma(I)$ is a minimal model of $P$.

Proof. By the fact that for $P$ the orbits of $G$ constitute a well-arranged partition of $R$, (1) holds. By Lemma 4.1 and the above fact again, (2) holds. □

Theorem 4.3. If $I$ is a minimal model of $P$, then there exist a minimal model $I_G$ of $G(P)$ and $\sigma \in G$ such that $I = \sigma(I_G)$; if $I_G$ is a minimal model of $G(P)$, then for any $\sigma \in G$, $I = \sigma(I_G)$ is a minimal model of $P$.

Proof. Assume that $I$ is a minimal model of $P$. For $r_i \in I$, let $O_i$ be the orbit of $G$ containing $r_i$, and $r_i'$ the representative of $O_i$ in the procedure to form $G(P)$. By Lemma 4.1, $O_i \cap I$ consists of one element. Let $\sigma = \prod_{r_i \in I} (r_i, r_i')$. Then, $\sigma \in G$, and $\sigma(I)$ is a minimal model of $G(P)$. 

If $I_G$ is a minimal model of $G(P)$, then according to Lemmas 4.2 (1) and 4.2(2), for any $\sigma \in G$, $\sigma(I_G)$ is a minimal model of $P$. □

If we use $MM_S$ to denote the set of all the minimal models of a clause set $S$, then Theorem 4.3, together with Lemma 4.2, says

$$G(MM_P) = MM_P = G(MM_{G(P)}).$$

The actions of the permutations in $G$ on all minimal models of $G(P)$ lead to all minimal models of $P$. To get the minimal models of $P$, we only need substitute atoms in the minimal models of $G(P)$ for those in the same orbits of $G$.

We turn to the completed models of $P$. Let $O_1$ and $C(1)$ be defined as in the procedure to derive $G(P)$, and $P_1 = \cup_{C \in P} C(1)$. If $I_1$ is a model of $\text{Comp}(P_1)$, then $I_1 \cap O_1$ contains at most one element. From this fact, the construction of each $C(1)$ in $P_1$ and the definition of $\text{Comp}(P)$, we have the following Lemmas.

**Lemma 4.4.** If $I_1$ is a model of $\text{Comp}(P_1)$, then there exists $\sigma_1 \in S_{O_1}$, such that $\sigma_1(I_1)$ is a model of $\text{Comp}(P)$.

**Lemma 4.5.** Suppose that $P$ is hierarchical, $I$ is a model of $\text{Comp}(P)$, and $O$ is an orbit of $G$. For two different $r_1, r_2 \in O$, if $r_1 \in I$ then $r_2 /\in I$.

Proof. If this is not the case, we prove that $O_i$ are all the orbits of $G$ such that $O_i \cap I$ contains more than one element. Choose an $r \in \cup_i (O_i \cap I)$ with the minimal level. Then there exist at least one $C \in P$ whose head is $r$, two different $r_1, r_2 \in M_C$, $r_1$ and $r_2$ are in the same orbit of $G$, and $r_1, r_2 \in I$. However, the level of $r_1$ is smaller than that of $r$. This is in contradiction with the fact that $r$ is the minimal level in $\cup_i (O_i \cap I)$. □

**Theorem 4.6.** (1) If $I_G$ is a model of $\text{Comp}(G(P))$, then there exists $\sigma \in G$ such that $\sigma(I_G)$ is a model of $\text{Comp}(P)$; (2) if $P$ is hierarchical and $G(P)$ keeps heads, then $\text{Comp}(P)$ and $\text{Comp}(G(P))$ are logically equivalent.

Proof. (1) By Lemma 4.4 we get the result as required. (2) Similar to the proof of (1), from the fact that $G(P)$ keeps heads, it follows that that if $I_G$ is a model of $\text{Comp}(G(P))$, then $I_G$ is a model of $\text{Comp}(P)$. On the contrary, by Lemma 4.5, if $I$ is a model of $\text{Comp}(P)$, then it is a model of $\text{Comp}(G(P))$. □

If we use $\text{Mod}(S)$ to stand for the set of all the models of a formula set $S$, then Theorem 4.6 (1) can be rewritten as

$$\text{Mod}(\text{Comp}(G(P))) \subseteq G(\text{Mod}(\text{Comp}(P))).$$

Up to some permutations in $G$ the completed models of $G(P)$ are completed models of $P$. As a corollary, $\text{Comp}(P)$ is consistent if $\text{Comp}(G(P))$ is consistent.

Generally, the converse of Theorem 4.6 (1) does not hold. Namely

$$G(\text{Mod}(\text{Comp}(P))) \not\subseteq \text{Mod}(\text{Comp}(G(P)))$$

is possible. However, for the (sequential) disjunction-invariant permutation group of $P$, the converse of Theorem 4.6 (1) holds, and Theorem 4.6 (2) holds for stratified programs.\[10,20\]

In the definition of a $G$-reduced form of $P$, if we neglect the number of occurrences of $\neg B_w$ in $B$, and define

$$C(k) = A \land B_{O_k} \rightarrow r_k$$

when $B_w \in O_k$, then since minimal models are syntax-free, Theorem 4.3 still holds. Theorem 4.6 (2) holds as well, since the number of occurrences of $\neg B_w$ in $B$ is zero when $P$ is hierarchical. But Theorem 4.6 (1) does not hold.\[10,20\]

5 Applications

We show how $G$ can be used in the assumption of negative information and some other classic model or procedural semantics. Our basic idea is to utilize the $G$-reduced form $G(P)$ as the underlying program. We first describe a rule to tackle negative information termed G-CWA by combining $G$ into the GCWA. Then we define G-definite, G-hierarchical, and G-stratified programs by combining $G$ into the definitions of definite, hierarchical, and stratified programs, and extend the corresponding semantics to them, respectively.

5.1 G-CWA

Assume that $O_i$ $(i = 1, \ldots, m)$ are all the orbits of $G$. For each $O_i$, let $O_i^+$ be a set consisting of an arbitrarily chosen element of $O_i$, and $O_i^- = O_i - O_i^+$. In the sequel, $r$ denotes an atom, and a disjunction means a disjunction of finitely many atoms. Let

$$NF_1 = \{ \neg \sigma \mid \exists O_i : r \in O_i^- \},$$
$$NF_2 = \{ \neg \sigma \mid \exists O_i : r \in O_i^+, \forall \text{ disjunction} \}$$
$$D : P \models D \lor \neg r = P \models D.$$

Let $G$-CWA($P$) = $P \cup NF_1 \cup NF_2$. $\neg r$ is said to be derivable from $P$ under the G-CWA (w.r.t. $O_i^+$) if $G$-CWA($P$) \models \neg r.$

**Theorem 5.1.1.** The G-CWA preserves consistency. That is, G-CWA($P$) is consistent.
Proof. In the procedure to form \( G(P) \), choose the atoms in \( O_i^+ \) as the representatives. By Theorem 4.3, \( G(P) \) is consistent. Let \( I_G \) be a minimal model of \( G(P) \). According to Theorem 4.3, \( I_G \) is a minimal model of \( P \). Moreover, \( I_G \) is a model of \( NF_1 \) and \( NF_2 \).

**Lemma 5.1.2.** \( P \models r \iff G(P) \models r \) and \( r \in \text{Fix}(G) \).

Proof. If \( P \not\models r \), there exists a minimal model \( I \) of \( P \), such that \( r \not\in I \). By Theorem 4.3, for some \( \sigma \in G \), \( \sigma(I) \) is a model of \( G(P) \). If \( r \in \text{Fix}(G) \), then \( r \not\in \sigma(I) \), \( G(P) \not\models r \).

If \( G(P) \not\models r \), there exists a minimal model \( I_G \) of \( G(P) \) such that \( r \not\in I_G \). By Theorem 4.3, \( I_G \) is a model of \( P \). So \( P \not\models r \); otherwise, if \( r \not\in \text{Fix}(G) \), let \( O_r \) be the orbit of \( G \) containing \( r \), \( r' \in O_r \) and \( r' \not\in r \). Let \( \sigma = (rr') \), and \( I_G \) a minimal model of \( G(P) \). Then \( \sigma \in G \), and by Theorem 4.3 again, \( \sigma(I_G) \) is a model of \( P \). But \( r \not\in \sigma(I_G) \). Thus \( P \not\models r \). \( \square \)

The next theorem shows that although new positive facts might be deduced by applying the G-CWA, we are able to recognize the situation rather easily.

**Theorem 5.1.3.** \( P \models r \iff G-CWA(P) \models r \) and \( r \in \text{Fix}(G) \).

Proof. If \( P \models r \), then \( G-CWA(P) \models r \), and by Lemma 5.1.2, \( r \in \text{Fix}(G) \).

If \( G-CWA(P) \models r \), let \( G(P) \) be that in the proof of Theorem 5.1.1. Then \( G(P) \cup NF_1 \cup NF_2 \models r \). If \( r \in \text{Fix}(G) \), then \( r \not\in \text{Fix}(G) \). Since \( R(NF_1) \cap R(G(P)) = 0 \), \( R(NF_1) \cap R(NF_2) = 0 \), \( G(P) \cup NF_2 \models r \). However, for any \( \neg r' \in NF_2 \), \( GCA(P) \models \neg r' \). So \( GCA(P) \models r \), and thus \( G(P) \models r \). Again from Lemma 5.1.2, it follows \( P \models r \). \( \square \)

Following the proof of Theorem 5.1.1 we know that the declarative description of G-CWA is \( \neg r \) is derivable from \( P \) under the G-CWA if \( r \) is in no minimal model of \( G(P) \). Indeed, the G-CWA is just running the GCWA on \( G(P) \). It is stronger from the declarative point of view. Also, in the procedure to formulate the G-CWA, if we choose each \( O_i^+ \) as an arbitrary non-empty subset of \( O_i \), all the results above still hold. Especially, when \( O_i^+ = O_i \) the G-CWA coincides with the GCWA. It also coincides with the CWA when the CWA is consistent. The G-CWA is thus a generalization of the GCWA. It makes the GCWA more powerful. Furthermore, we can immediately assume all the other atoms in the same orbit of \( G \) to be negative after choosing a representative. It is easier to infer negative information in this respect.

Until so far, we do not know yet the relations between the G-CWA and the careful CWA. We wonder if the G-CWA might be covered by the careful CWA. However, this does not seem theoretically important, since we can also take \( G \) into account when performing the careful CWA. The G-CWA has another advantage that one can modify the negative facts quite easily when his mind on the world is changed. In the careful CWA, it is difficult to conduct such changes directly from the derived negative facts. One drawback of the G-CWA is that when there are no non-trivial permutation groups associated with \( P \), the G-CWA is exactly the GCWA.

### 5.2 G-Definite, G-Hierarchical and G-Stratified Programs

\( P \) is said to be \( \text{G-definite} \) [\( \text{G-hierarchical}, \text{G-stratified} \)] if \( G(P) \) is definite [resp. hierarchical, stratified]. From the construction of \( G(P) \), we know that definite [hierarchical, stratified] programs are G-definite [resp. G-hierarchical, G-stratified], and G-definite and G-hierarchical programs are G-stratified.

#### 5.2.1 G-Definite Programs

We demonstrate that the SLD-resolution can be used for G-definite programs and definite goals without losing the soundness and completeness.

Let \( r \) be an atom. If \( P \) is G-definite, then \( G(P) \) is definite. Let \( I_G \) be the least model of \( G(P) \).

**Lemma 5.2.1.** Let \( P \) be G-definite. The following is the same: (1) \( \{ \bar{r} \mid P \models \bar{r} \} \); (2) \( I_G \cap \text{Fix}(G); \) (3) \( \{ \bar{r} \mid \text{there is an SLD-refutation from } G(P) \cup \{ \bar{r} \rightarrow \} \cap \text{Fix}(G) \} \).

Proof. Immediate from Lemma 5.1.2. \( \square \)

By Lemma 5.2.1, we have Theorem 5.2.2.

**Theorem 5.2.2.** Let \( P \) be G-definite, and \( Q \) a definite goal. \( P \models \neg Q \) iff there exists an SLD-refutation from \( G(P) \cup \{ Q \} \) and \( R(Q) \subseteq \text{Fix}(G) \).

This theorem indicates that the SLD-resolution is sound (we need to test whether \( R(Q) \subseteq \text{Fix}(G) \) is in advance) and complete for G-definite programs and definite goals. On the other hand, if \( P \) is definite, the results of SLD-resolution from \( P \cup \{ Q \} \) coincide with those from \( G(P) \cup \{ Q \} \). As a matter of fact, in this case the least model of \( G(P) \) is exactly that of \( P \).

Remark that the fair SLD-resolution from \( G(P) \) may be used to deal with negative information. When \( P \) is definite and \( G \) is the (sequential) disjunction-invariant permutation group of \( P \), it is
the implementation of the negation as a failure rule from \( P \). Please see [19] for the details.

### 5.2.2 G-Hierarchical and G-Stratified Programs

We show that the SLDNF-resolution is complete for G-hierarchical programs in some sense. Furthermore, we extend the standard model semantics and the interpreter proposed in [4] to G-stratified programs.

Let \( H \) be the set of all atoms appearing in the heads of the clauses in \( P - P_h \). \( O_1, \ldots, O_h \) be all the orbits of \( G \) whose intersections with \( H \) are non-empty, and

\[
G_H = S_{O_i \cap H} \times \cdots \times S_{O_i \cap H} \times I_H.
\]

where \( I_H \) is the trivial permutation group on \( H \).

If \( Q = \bigwedge_i L_i \rightarrow \) is a normal goal, we use \( G_H(Q) \) to denote the normal goal \( \bigwedge_i \forall x \in G_H \sigma(L_i) \rightarrow \).

Assume that \( G(P) \) keeps heads. We say that \( Q \) is correct w.r.t. \( P \) if \( \text{Comp}(G(P)) \models \neg G_H(Q) \). \( Q \) is said to be computed w.r.t. \( P \) if there exists a safe computing rule if there exists an SLDNF-refutation from \( G(P) \cup \{ G_H(Q) \} \).

It is reasonable to define the correctness by using \( G(P) \) in this way. Indeed, only one atom in each orbit of \( G \) needs to be kept, the others can be treated as negative. On the other hand, the atoms in \( O_i \cap H \) have equal status. Hence, they should have the same opportunity to be heads. Actually, \( Q \) is correct w.r.t. \( P \) iff for any \( G(P) \) keeping heads, \( \text{Comp}(G(P)) \models \neg Q \). \( Q \) is computed w.r.t. \( P \) iff for an arbitrary such as \( G(P) \), there exists an SLDNF-refutation from \( G(P) \cup \{ Q \} \).

Remark that \( G_H \) is trivial when \( P \) is hierarchical. Therefore, by Theorem 4.6(2), the definitions of \( Q \) being correct and being computed w.r.t. \( P \) coincide with the usual ones[5].

### Theorem 5.2.3. Suppose that \( P \) is G-hierarchical, and \( Q \) is a normal goal. \( Q \) is correct w.r.t. \( P \) iff \( Q \) is computed w.r.t. \( P \).

Proof. From the completeness result of SLDNF-resolution for hierarchical programs, it follows that \( \text{Comp}(G(P)) \models \neg G_H(Q) \) iff there exists an SLDNF-refutation from \( G(P) \cup \{ G_H(Q) \} \).

G-hierarchical programs may allow some recursion. From the above remark, we know that Theorem 5.2.3 is indeed a generalization of Clark’s completeness result of the SLDNF-resolution for hierarchical programs. Moreover, some other completeness results, e.g., the result in [10] can also be treated in a similar manner.

We turn to G-stratified programs. By Theorem 4.6 (1), we have the following theorem, which extends the result that the completion of a stratified program is consistent[4]. Hence, the completion-based proposal (The negation of an atom is inferred if it follows from the completion of the program) to tackle negative information is suitable for G-stratified programs. Its expressive power is thus enhanced.

### Theorem 5.2.4. If \( P \) is G-stratified, then \( \text{Comp}(P) \) is consistent.

Let \( P \) be G-stratified, and \( G(P) \) keep heads. We call the standard models of \( \sigma(G(P)) \) (for any \( \sigma \in G_H \)) the standard models of \( P \).

Similarly, it is reasonable to utilize these models as the intended meaning. This definition extends the usual standard model semantics for stratified programs. When \( P \) is stratified, \( G_H \) is trivial, and by Theorem 4.3 and the procedure to get \( G(P) \) which keeps heads, we can prove that the standard model of \( P \) is exactly that of \( G(P) \).

Note that in the predicate case locally stratified programs[7] can be similarly treated though we have to face infinite programs.

For an atom \( r \), we say \( P \models_{SM} r \) if \( r \) is in the intersection of the standard models of \( P \), \( P \models_{SM} \neg r \) if \( r \) is not in the union of the standard models of \( P \).

Although a G-stratified program may not have the unique standard model, the interpreter described in [4] does apply. Also, we need not apply the interpreter to all the G-reduced forms of \( P \) keeping heads. We only need to fix one of them.

### Theorem 5.2.5. Suppose that \( P \) is G-stratified, \( G(P) \) keeps heads, and \( L \) is a literal. \( P \models_{SM} L \) iff for any \( \sigma \in G_H \), \( G(P) \models_{SM} \sigma(L) \).

Proof. If \( L = \neg r \), where \( r \) is an atom, then \( P \models_{SM} \neg r \), iff for any \( \sigma \in G_H \), \( r \) is not in the standard model of \( \sigma(G(P)) \), iff for any \( \sigma \in G_H \), \( \sigma^{-1} r \) is not in the standard model of \( G(P) \), iff for any \( \sigma \in G_H \), \( G(P) \models_{SM} \neg \sigma(r) \).

The conclusion when \( L = r \) can be proved analogously.

Remark that the results in [9, 10, 22] do not imply Theorem 5.2.3 and 5.2.4, or vice versa. The perfect model coincides with the standard model for a stratified program. It was shown in [23] that the perfect model semantics can be defined for stratified programs up to redundancy. We point out that the notion of being G-stratified is not covered by the notion of being stratified up to redundancy, or vice versa[19, 20].

Actually, whether our results are covered by
some already existing work is not so important. The key point is that a paradigm in logic programming is proposed. As we have already stated, we can also consider combining G into that semantics. For example, we can analogously define G-call-consistent and G-locally stratified programs, etc. based on the G-reduced form G(P), and then think of extending the corresponding semantics to these more general programs. Here, we do not go into the details.

6 Generalizations to the First-Order Case

Hereafter, we denote by r_i(t_i) an atom in which t_i is an n-tuple of terms if r_i is an n-ary predicate symbol. Let R and R(M) be the sets of all the predicate symbols appearing in P and an atom set M, respectively. In the first-order predicate case, we say that for P a partition of O_1, . . . , O_m of R is well-arranged, if for any C in P the following condition holds.

If some predicate symbol occurring in R(M^+_C) is in a proper O_i, then there exist r_2 and two different r_1(t_1), r_2(t_2) ∈ M^+_C, such that r_1, r_2 ∈ O_j and t_1 = t_2; otherwise, for an arbitrary r_1(t_1) ∈ M^+_C, if r_1 ∈ O_i then for any r_2 ∈ O_j, r_2(t_1) ∈ M^+_C.

The permutation group G associated with P and the G-reduced form G(P) of P (keeping heads) can then be defined analogously. Notice that a clause set has minimal models, which are strictly defined in [24].

Now, let I be an interpretation, D_I be the domain of I, and R_I the predicate symbol assignment of I. For an n-tuple d ∈ D_I^n and σ ∈ G, let

\[ I(d) = \{ R_I(r)(d) | R_I(r)(d) \in I \}, \]
\[ \sigma(I_d) = \{ R_I(\sigma(r))(d) | R_I(r)(d) \in I \}. \]

Then Theorems 4.3 and 4.6 can be rephrased as the following two theorems. Since their proofs are not essentially different, we present no longer the details.

**Theorem 6.1.** If I is a minimal model of P, then there exist a minimal model I_G of G(P) and σ_G ∈ G such that I = \bigcup \sigma_G(I_G(d)): if I_G is a minimal model of G(P), then for any σ ∈ G, I = \bigcup \sigma_G(I_G(d)) is a minimal model of P.

**Theorem 6.2.** (1) If I_G is a model of Comp(G(P)), then there exist σ_G ∈ G such that \bigcup \sigma_G(I_G(d)) is a model of Comp(P); (2) if P is hierarchical and G(P) keeps heads, then Comp(P) and Comp(G(P)) are logically equivalent.

Accordingly, all the results in Section 5 hold in the first-order predicate case. To avoid floundering, in Theorem 5.2.3 we usually require that \( P \cup \{ Q \} \) is allowed[39]. If \( P \cup \{ Q \} \) is allowed, then \( G(P) \cup \{ Q \} \) is obviously allowed. However, for the definition of being allowed given by Lloyd et al.[8], this is not true even when P is hierarchical. Fortunately, if we further require that G(P) keeps heads, this claim holds again.

7 Concluding Remarks

One shortcoming of the permutation groups utilized in this paper is that in some cases they are trivial. It is impossible to expect that every program associates with a non-trivial symmetric structure. However, there do exist more general ones. A program induces a number of symmetric structures, which can be roughly classified into the syntactic and semantic ones. Besides the (sequential) disjunction-invariant permutation group G of P, for instance,

\[ G_1 = \{ \sigma \in S_R | \forall C \in P, \exists C' \in P : \sigma(M_C) = M_{C'} \} \]

is another syntactic permutation group induced by P, and \( G \subseteq G_1 \) (i.e., G is a subgroup of G_1). The following are some examples belonging to the category of semantic permutation groups induced by P.

\[ G_2 = \{ \sigma \in S_R | \forall C \in P, \exists C' \in P : \sigma(C) \]

is logically equivalent to \( C' \},

\[ G_3 = \{ \sigma \in S_R | I \text{ is a model of } P \Rightarrow \sigma(I) \text{ is a model of } P \}, \]

\[ G_4 = \{ \sigma \in S_R | I \text{ is a minimal model of } P \Rightarrow \sigma(I) \}, \]

\[ G_5 = \{ \sigma \in S_R | \text{Comp}(P) \text{ is logically equivalent to Comp}(\sigma(P)) \}. \]

These permutation groups keep some syntactic or semantic properties invariant, and G ≤ G_1 ≤ G_2 ≤ G_3 ≤ G_4. The structures of G_i (i ≥ 1) are more complicated than G. It is interesting to explore the behaviors of a program under them or their combinations.

We need not limit ourselves to the symmetric structures on predicate symbols. We can also consider, e.g., those on function symbols and constants ([26] has applied this kind of symmetries to theorem proving), and consider the actions of them or their combinations.

If we similarly define a permutation group on function symbols and the corresponding reduced forms, generally Theorems 4.3 and 4.6 do not hold anymore.[19,20] However, if P is a ground clause set, all the similar conclusions hold.
The third criterion to evaluate a semantics is the computational issue. We did not pay more attention to this aspect in this paper. Instead, we concentrated on the expressive power and declarative ability of semantics. The introduction of symmetric structures in logic programs may make some computation procedures simpler[19]. How difficult it is to find a permutation group associated with a program remains unsolved. We need also analyze some bigger real-world examples to see the practical usefulness of our approach.

Several authors have taken advantage of symmetric structures to simplify the resolution-based methods in first-order theorem proving[25, 26]. The permutation groups utilized in this paper is generally smaller than those used in [25]. This is quite understandable since basically in theorem proving one cares only about the preservation of the satisfiability rather than the preservation of some further properties of a clause set. Obviously, logic programming is more complicated from the viewpoint that the computations are often involved with some semantics beyond the classical inference rules.

References


Jin-Zhao Wu was born in 1965. He obtained his Ph.D. degree in 1994 from the Institute of Systems Science, the Chinese Academy of Sciences. From 1994 to 1999 he was a postdoctoral and research scientist in Peking University and MaxPlanck Institute of Computer Science. Since 2000 he has been working on the Faculty of Mathematics and Computer Science, University of Mannheim.

Harald Fecher was born in 1972. He obtained his Ph.D. degree in 2003 from the Faculty of Mathematics and Computer Science, University of Mannheim. Since 2004 he has been research scientist in the Faculty of Computer Science, University of Kiel.